CSCI 501/701 – (Advanced) Software Principles and Practice
Assignment #2

Due Date: Friday, September 6 at 23:59

Submit your solutions to Moodle when completed.

DESCRIPTION

For this assignment, you will be creating two implementations of the Stack ADT, using the approaches that we have discussed in class:

Part 1:

- Create a new C project and provide the implementation of the integer stack component using the array-based approach from the slides. Here, you are provided the interface information (i.e., function signatures) in the stack[1].h file, along with the underlying data representation. **Download stack[1].h file and rename it to stack.h file**.
- Create another file called **stack_test.c**, which creates one or more instances of the stack you just defined, and thoroughly tests each of the functions.
- **Submit** your **stack.c** and **stack_test.c** files to "**A2 – Part 1 – Array based stack**" Moodle.

Part 2:

- Create a second C project, copy over your stack_test.c file from Part 1, and download and add the second stack[2].h file from Moodle to the project. **Before starting working please also rename stack[2].h file to stack.h file**. This version of stack.h uses a linked-list representation for stack as discussed in the lesson.
- Create a new **stack.c** implementation file, which uses the underlying linked-list representation given in **stack.h**.
- Use the **stack_test.c** testing file from the previous part to test your new implementation – you should not have to make any changes to this file for things to compile and work!
- Submit your new **stack.c** file from this part to "**A2 – Part 2 – Linked list stack**" in Moodle.

CONSIDERATIONS

- For Part 1, be sure to first make sure you have enough space on your array to perform a push. If you don't have enough space, create a new array of double the size, and copy over the previous values. (Don't forget to free the old array once you are done!)
- For Part 2, make sure you properly manage old and new nodes. Be sure to malloc a new node when pushing, and be sure to free the old top node when popping.
- For both parts, if you try popping off an empty stack, and error message should be printed to the screen, and the program should not crash or freeze – in other words, don't try to access something that you shouldn't in the array, or reference a non-existing node. You should return the value INT_MIN in this case (which is defined in the limits.h library).
- For both implementations, use malloc/calloc and free to dynamically create (and destroy) the non-primitive items (i.e., arrays and nodes) that are referenced from the given structs. For Part 2,

don’t forget that you need to free all nodes in the linked-list when it is cleared or destroyed, not just the first node!